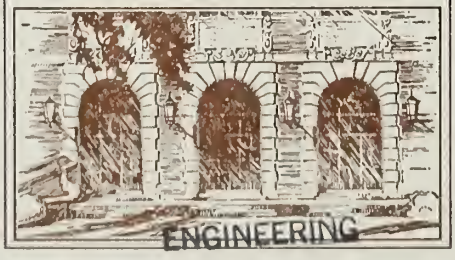


Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document No. 22

An Investigation Into Information Retrieval Utilizing ILLIAC IV

By

B. Wittman
Allairion Company

November 30, 1971

CAC Document No. 22

An Investigation Into Information Retrieval Utilizing ILLIAC IV

by

B. Wittman
Information Counselor
Allairion Company

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

November 30, 1971

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04 72-C-0001.

Abstract

This document presents an investigation into the effective use of the ILLIAC IV for information retrieval. It notes the lack of parallelism in the IO structure and finds two orders of magnitude gain over serial machines when using serial processing. It recommends serial access methods and linearized structures where such structures are possible. It also recommends changing the ILLIAC IV IO structure. Further, the document demonstrates a linearization technique for graphs.

TABLE OF CONTENTS

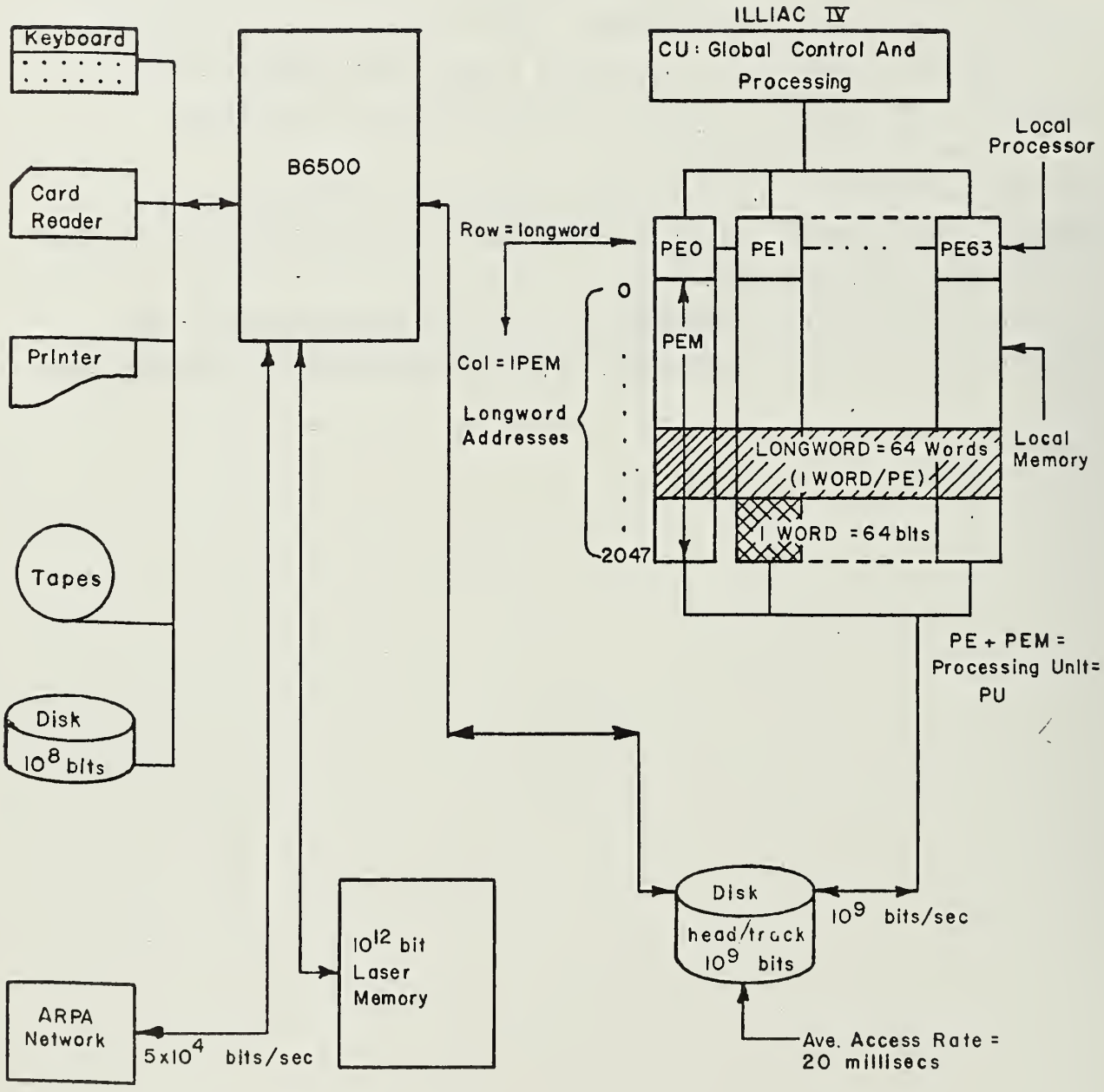
Introduction

This report is written to emphasize results. Thus, conclusions are generally first stated, then supported. Gory support details are relegated to an appendix, denoted as a reference by a superscript capital letter representing the appendix enclosed within parentheses, for example:

"...can look quite complex.$^{(D)}$"

Footnotes and figures are placed within the text to avoid annoying page-flipping. Bibliographic references are superscript parenthesized numbers and may include supplementary comments.

Figure 1, following immediately, is not referenced directly in the text, but is included as a piece of orientation information for completeness.



(4)

ILLIAC IV Hardware Relevant to Information Retrieval

FIGURE 1

Summary

The ILLIAC IV machine has been investigated with an eye toward information retrieval. The results were discouraging at that time and are not appreciably better now. Because refinements to serial data organizations are made to allow simulations of content-addressable-memories, it was to be expected that the ILLIAC IV, which can simulate a sixty-four element content-addressable-memory, would be precisely matched to information processing and data management problems. Alas, the awkward input-output structure of the ILLIAC IV destroys this hope. It is frustrating to be led ineluctably to the conclusion that a machine with such power and speed as the ILLIAC IV is unsuitably organized to attack the single most important problem facing everybody everywhere; yet it does seem to be the case that the ILLIAC IV is in fact not the answer to information retrieval problems in general.

There do exist two classes of information retrieval problems to which the ILLIAC IV is suited: those problems which are analyses, breakdowns, summaries or other reductive calculations including all or most of the associated file, and problems whose data structures can be linearized to suit them. These may include rather complicated network structures, but structural strictures are stringent with regard to allowable flexibility and future utility. These structures must be essentially free of loops, for the price of every loop included in the final structure may be a very heavy retrieval-time cost.

A Voice from the Past

D. H. Lawrie authored two documents [1] concerning information retrieval using the ILLIAC IV. One of his general conclusions was that for files which had one million entries (a number we shall continue to use for illustration), no type of index table (hash, single-linked or tree) could be kept in core; consequently much of the savings of using such an organization to reduce the number of data accesses was lost through the offsetting need for index table accesses. "Use of binary trees affects

(sic) a modest gain..."[2] observed Lawrie, but his final conclusion was considerably more discouraging: "...we don't want to use the ILLIAC IV for information retrieval unless we have to as we don't achieve any great speed up over a serial machine...".[3]

Access Methods

There are two types of people in the world: those who divide everything into two groups and those who do not. Those who do will speak of serial access as opposed to all other access modes. The author tends to avoid this Aristotelian viewpoint, but this tendency to dichotomize turns out to be utilitarian here, though we would rather speak of static versus dynamic access structures.

Static access structures are those in which the elements of the structure are always accessed in the same order. Serial access is the prime (and possibly only) example of such a structure, but it is not its serial quality but its static quality which will concern us. Since the order of a serial file is really not relevant in the most general sense of serial, any ordering can be selected. This ordering is then physically realized and becomes the single order in which the elements are accessed. This is a natural "staticization" of the file, that is, it imposes a structure on the file which allows static access to be meaningful.

Dynamic access structures, by contrast, are structures in which the elements are not accessed in the same order every time the file is used, but may be accessed in any order depending on the problem. Dynamic access methods depend (at least in all their physical realizations) on pointers to direct the dynamic trace through the file elements.

Serial access is the oldest, simplest and most straightforward access method known to data processing. However, of necessity (being static), access to the file means access to the whole file. This is wasteful for applications wherein only a small portion of the file is (pre)known to be relevant. First attempts to overcome this problem centered on batching of requests to render larger portions of the file relevant, thus reducing the percentage waste of a single serial access. This refinement, however,

remained too slow for many applications. Consequently a whole new approach was sought. Several new non-serial access methods were discovered, though it was not really their non-seriality which was so important; it was their dynamic access structure: the elements could be accessed in more than one order, which also implies that some need not be accessed at all. These methods included index-sequential, direct (random, hashed), secondary indexing, multi-list, cellular multi-list, file inversion, etc. Regardless of the theory of each of these techniques, they all relied on keys and pointers to achieve dynamic access structure. More complex problems were now capable of being represented too, since these new access structures could be utilized to carry data structure; that is, to show a relationship between data components.

The immense speed achieved by the parallel structure of the ILLIAC IV should be equally applicable to both static and dynamic access structures to achieve equally excellent results; to the extent that the ILLIAC IV is organized in parallel this observation is true. It is critical to note, however, that the parallellism of the PE's (Processing Elements) is not maintained in the input-output (IO) structure which causes a minimum of sixteen PE's to receive data from a single source. Thus the ILLIAC IV is particularly inept (in comparison with its potential) at dealing with keys and pointers rendering all previous dynamically structured access methods and accompanying expertise in them largely useless and irrelevant.

Either an entirely new concept of file organization must be discovered or the use of the ILLIAC IV for information retrieval must be suitably restricted if optimum efficacy is to be achieved. Considering that a completely new file organization must be invented merely to reap the benefits of serial access on the ILLIAC IV, [4] it seems unwise to speak of revolutionary file structures in the immediate future. Structural theoreticians may provide some assistance as familiarity with the ILLIAC IV grows, but little relief is apparent in the near future.

The Problem of Non-Static Access on the ILLIAC IV

The IO structure of the ILLIAC IV is such that at least sixteen (16) PE's must be loaded from a single data block. A dynamic access scheme implies keys and direct pointers to desirable data blocks to minimize access to unneeded data, thereby holding waste to a minimum. Though each PE in a set of 16 may compute independently which block it should access next, only one of the 16 blocks so determined can actually be accessed at once; thus, the waste factor approaches 15:1 (it being possible, though relatively infrequent, that more than one PE desires the same block). Further, the effective IO rate is slowed by transmitting only 16 PEM's worth of data at a time instead of the full 64 PEM's possible on each data transfer.

While it may be argued that a waste of 15:1 is not very large compared to the waste from serial access for applications which require access to a miniscule portion of the file, it should still be noted that the inter-PE routing and swapping of the index tables in and out of core contributes significantly to the access time, and that any waste on a machine as costly as the ILLIAC IV is cause for serious revaluation.

Generalizations

It is not only information retrieval which suffers from the peculiar IO structure; multi-programming and time-sharing suffer similarly since data paging is essentially identical to the non-static access problem: pages are stored and retrieved by pointers and keys, and blocks which load 16 PEM's at one time simply cannot be constructed as static entities for this type of application.

In general we may consider any process to consist of three parts:

1. Process definition
2. Core-resident data
3. Peripheral-resident data

Under certain conditions (whose investigation is a topic in its own right and well outside the concerns of this report) the process definition can be described in such a manner as to exploit parallel processing. So long as the RGX's (index registers) exist the process can be operated in parallel, even though the core-resident data are different for each PE causing the

need to access different relative locations in the PEM's. But, if the sequence of data access from the peripherals is dynamic rather than static, the data cannot be preformatted to utilize parallellism with guaranteed efficacy and the process is doomed to be relatively inefficient.

The general conclusion that we may draw is that any problem to be implemented with optimum efficacy on the ILLIAC IV must have an access structure which is static, at least within a 16-PEM block.

The following sections will deal with demonstrating the actual efficiency differences between static and dynamic access structures, and attempts to force apparently non-static data structures into a static access structure.

Assumptions and Definitions

In the following sections, formulae will be cited (and developed in the Appendices) and typical numbers will be demonstrated in them. The variables and their sample values which are used in these formulae are defined below. The meaning of "[x]" is "the integral part of x", whereas $[x]^+$ means "x rounded up".

Variables and values:

n = number of records in the file (one million, 10^6)

b = buffer size per PEM in bytes (128 words = 2^{10} bytes)

a = average buffer-load time (45 milliseconds maximum)

d = size of data record in bytes (1000 = $10^3 \simeq 2^{10}$)

x = size of index record in bytes (16: value+pointer, 1 word each)

t = execution time of one loop in computation (5 microseconds)

e = number of PEM's appearing as one group (16 or 64)

p = size of value list, number of entries in list (100)

Derived variables:

$m_d = [b/d]$ = number of data records per buffer per PEM

$m_x = [b/x]$ = number of index records per buffer per PEM

$M_d = m_d * e$ = number of data records per physical block

$M_x = m_x * e$ = number of index records per physical block

$n_o = [n/M_d]^+$ = number of physical data blocks constituting the file

$n_i = [n_{i-1}/M_x]^+$ = number of physical index blocks at level i

f = smallest number such that $(M_x)^f \geq n_o$ = maximum level of index

Compute-Time Considerations

The speed with which the ILLIAC IV calculates suggest that the limiting factor in information retrieval will most likely be the IO rate. This is substantiated by the following computations.

The average time, T, to check a buffer full of records against a list of desired values (using a binary search process) is:(A)

$$T = [b/d]*(1-\frac{p}{2n})*([\log_2 p]+1)*t.$$

If we presume some fixed value for T, the amount of time each PE has to complete its computations, then the larger [b/d] and t are, the smaller p (list size) will be to avoid compute-boundedness. Assuming

t = 5 micro-seconds and [b/d] = 10 records-per-buffer

both reasonable and possibly generous figures, then:

$$T = 50 * ([\log_2 p] + 1) *(1 - \frac{p}{2n}) \text{ micro-seconds.}$$

Since $0 \leq p \leq n$ then $0 \leq T \leq 50*([\log_2 p]+1)$.

For a file of one million records, since $(2^{19} \leq 10^6 \leq 2^{20})$ then

$0 \leq T \leq 1$ milli-second.

Thus if the time available for computation on a million-record file were as little as 1 millisecond per buffer, then the whole set of unique identifiers for all the file records could be searched: $p=n=10^6$ is the size of the list which generates a compute-bound condition. Since the identifiers must be at least 20 bits in length, there is not sufficient room in core to hold enough entires to cause compute-boundedness; we run out of space before we run out of time.

Thus unless the available compute time per buffer is very small the process will be IO bound. One-half millisecond is sufficient to process $2^{11} = 2048$ entries. (Reference Figure 2 following). This is approximately 2% of the file which is more than we should ever seek in this manner. Since this many entries will not take more than half of core, it is a reasonable assumption that the process will always be compute-bound and that core space is sufficient.*

*NOTE: Apparently, the current 4096 longwords are expected to be expanded to 8192 longwords. This allows, for instance, 2048 words for buffer space, 2048 words scratch space, and 4096 * 64 words instruction space. This should be plenty.

The following graph demonstrates a curve showing the time needed per buffer to search a list whose size is a fraction of the number of records (expressed in negative powers of 2); on semi-log paper it would be a straight line.

(Note the method has the usual peculiar binary search characteristic: searching for 1% of the total number of records takes about 2/3 of the time to search through all 100% of them.)

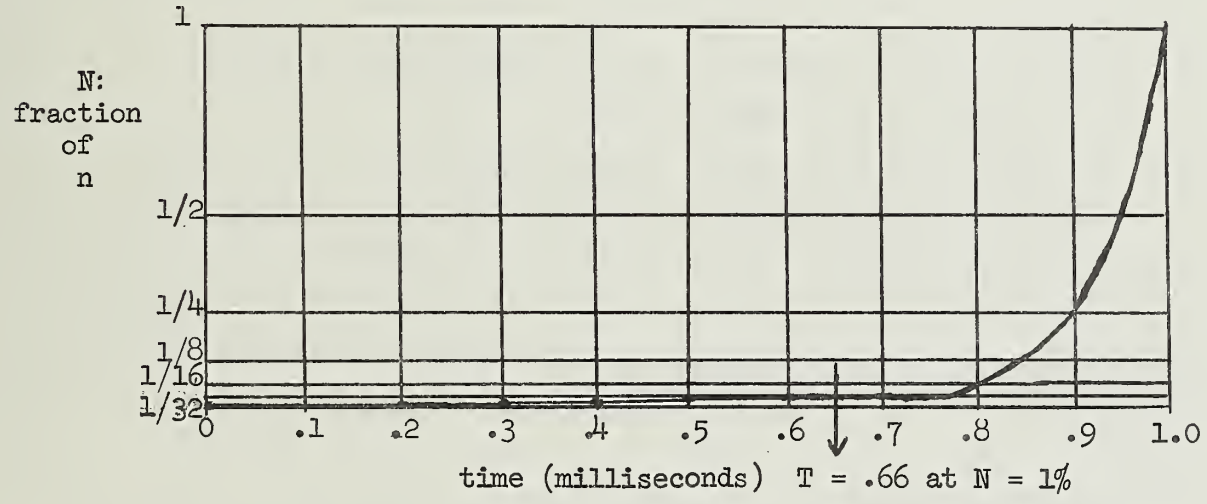


FIGURE 2

Serial Access

The ILLIAC IV can perform a serial search utilizing its parallellism to the fullest (since serial access is inherently static access). Since a PE appears to be of approximately the same speed[5] as a CDC-6600 or an IBM-360/75, we can expect a gain in speed equal to the parallelism of the ILLIAC IV, a factor of 64. Further promise lies in the fact that the speed of the ILLIAC disc is approximately 3 times faster than currently popular discs, such as the IBM 2311 disc pack or the RCA 564.

Serial pass time is:

$T = n_o * a = [n/([b/d]*64)]*a = 2^{20}/(2^{10}/2^{10}*2^6)*a = 2^{14}*a.$

As noted earlier, a, being dependent on latency, buffering techniques, IO-control communication times, and data transfer rates, is difficult to estimate. Assuming the worst, only one buffer, the slowest processing time will be achieved. We assume the worst case on everything:

a = 45 milliseconds = 40 mils/revolution[6] + 5 mils event-time.[7]

Under this gruesome set of assumptions, file pass time is $2^{14}*45$ mils, or

T = 12 minutes.

The file in this sample is about 60 times the size of a tape file, hence

T = 12 seconds for a tape-size file.

Using the suggested 6 buffers [8] or a buffer 6 times as large:

T = 2 seconds for a tape-size file.

This is a viable result - in fact an impressive one.

An ordinary machine would lose a speed factor of 64 from lack of parallellism and another factor of 3 from disc differences, for a total slow-down factor of 192. Thus we see that the ILLIAC IV is 2 orders of magnitude faster than an ordinary machine of about equivalent electronic speed.

Index-Sequential Access

It has already been noted that adequate core space to hold enough unique identifiers to drive the ILLIAC IV into a compute-bounded state in searching for them would be absurdly large. Since IO-boundedness is the problem, we investigate methods of limiting IO. Following tradition and common sense we start with the simplest such method, index-sequential, a full description of which is contained in Appendix B. Basically, the

data is indexed using keys and pointers to point through index levels to the exact data block desired. Thus to satisfy a request for p records requires reading only p data records plus associated index records.

As we have already noted, each of a group of 16 PE's can determine which block it would like to have next, but only one of them, in general, can be satisfied by the next access. Consequently, the parallellism of the process is reduced from 64 to 4.

The formulae for computing the number of index blocks at a given level were displayed previously. Using them on our standard figures we find that only 2 levels of indexing are necessary; in fact since only one-sixteenth of the second-level index block is used, the file could be expanded by a factor of 16, to 16 million records, and still use only 2 levels of indexing. A file this size would use one-tenth of the laser memory.

Using the given sample values in the cited formulae, the computations are:

$M_d = m_d * e = [b/d]*16 = [2^{10}/2^{10}]*2^4 = 2^4$

$M_x = m_x * e = [b/x]*16 = [2^{10}/2^4]*2^4 = 2^{10}$

$n_o = [n/M_d]^+ = 2^{20}/2^4 = 2^{16}$

$n_1 = [n_o/M_x]^+ = 2^{16}/2^{10} = 2^6$

$n_2 = [n_1/M_x]^+ = [2^6/2^{10}] = [1/16] = 1$ (f=2).

The speed of index-sequential centers on minimizing the number of disc accesses made. To deliver a single record requested by its unique identifier requires only 3 disc accesses (one at each level) for a total time of only 3*45 = 135 milliseconds, but most of the machine is doing absolutely nothing at this time, and considering its hourly cost ($1000-$2000[9]) it will be expensive indeed to use so little of it. One million times as many records can be accessed in 2 minutes which is only 120 times as long. This is a 4 orders of magnitude loss, gross inefficiency.

To improve the efficiency requests may be batched. Assuming a batch of 100 requests (.1% of the file) which implies getting some extra usage out of the index accesses we can compute the total time needed. The level 2 index block must be accessed, this in turn will lead to 51 level 1 index accesses, and finally 100 data accesses.[C,10] Thus total access time is seen to be:

$T = ([1/4]^+ + [51/4]^+ + [100/4]^+)*45$ milliseconds = 1.35 seconds.

Again, using 6 buffers: T = .2 seconds

Using index-sequential we can achieve in .2 seconds what it would take us 2 seconds to achieve serially (and about 7 minutes on an ordinary machine). While it may very well be worthwhile to achieve a single order of magnitude speed-up over ordinary machines, we have already seen ways to achieve two orders of magnitude improvement. Consequently, though index-sequential may be of some utility, the cost/performance ratio of the ILLIAC IV suggests the need to concentrate on the area of largest gain first.

Other Non-Static Access Methods

Since other non-serial (non-static) access methods deal with multiple keys and pointers which in general refer to more than one record, we can visualize a rapid deterioration in efficiency in attempting to employ such structures for the ILLIAC IV. With only a shallow margin of speed-up, at best, realized from index-sequential, the value of multi-keying is highly questionable.

While index-sequential may be useful as an alternate method of access to a file processed mostly serially, the multi-keyed organizations may be useful for the extension of capability to represent certain data structures, whose pursuit seems ill-advised, at least until the larger potential gains are realized.

Considering the complexity of the subject, no figures will be introduced here in support of this argument. It is clear from qualitative considerations alone that complex keying schemes are not a very fruitful approach to information retrieval on the ILLIAC IV.

New Data Organizations

It has already been noted[4] that the records of a file intended for ILLIAC IV serial processing must be stored in a new and very unusual way to cause them to appear in core so that each record is contained in one PEM instead of split across one or more PEM's. This document assumes that such an organization is feasible and well understood. There are, of course, practical and implementational problems associated with this new structure, and the problem may turn out to be more complicated than currently imagined, but it seems safe to assume that these potential complications will not be in philosophy but of a more tractable order of magnitude, and consequently this organization, and the process of creating it from standard files is taken to be known.

We have already determined, however, that any further revolutionary organization on the ILLIAC IV is not likely nearby, and that serial access is by far the most efficient method of file search on the ILLIAC IV in

comparison to other machines. It, therefore, is obvious that for maximum gain we should address ourselves to discovering just what abstract structures are vulnerable to linearization (serialization, staticization).

Networks

It is well known that trees can be linearized in several ways. Certain networks can also be linearized. The linearization process is, to some degree, dependent on the problem set for which the data will be used; hence, we might say that the network can be custom-linearized for a process.

Networks that can be linearized have no loops. They are directed graphs which, by repeating nodes, can be expanded into trees. Several such trees may be woven together if every node common to more than one tree has the same subtree in all such trees. (This condition is in fact more stringent than necessary. It is sufficient that the path from a given root to a given leaf, which can be expressed as an ordered n-tuple, does not violate an ordering defined by any other such path. This must be true for the entire collection of such paths.)

Once such a network is linearized it can be examined in a single file pass, that is, traced in a single file pass. If the network does contain loops (rings) it can only be partially linearized. Each pass over a loop requires re-passing part of the file. If the loops are infrequent and near the leaves of the expanded tree, the penalty of a second pass will be paid rarely. If the loops are frequent and nearer the root, they will often cause extra passes, and if they are interwoven at all they will cause many extra passes.

If the whole file is structured as a simple tree, a certain ordering is implied for the whole file. "All records at level i will precede records at level j for $i<j$" is such an ordering (with the understanding that records at the same level will be ordered in some fashion which preserves their relation to superior records). Such a file may seem atypical. In fact, the constraints imposed here, while sounding relatively mild, may leave the reader hard-pressed to think of a useful example for which such a file is á propos. It turns out that there is at least one commercial problem, the assembly problem, which fits this structure as though tailor-made.

The Assembly Problem

One of the problems which has been getting some attention recently is "parts explosion": determining what kind of components and how many of each are needed to manufacture a certain assembly (or subassembly). The sister problem is the determination of what can be done with a certain component and how many of them are needed to do it. Files to solve these problems are set up as bi-lateral networks involving two pointer chains: the explosion chain and the where-used chain. The graph appears bi-directional because it has forward and backward pointers, yet the application is such that only one pointer set is used at a time. Considering the pointer sets independently leads to two orderings, one (not surprisingly) being the reverse of the other. Consequently the explosion problem can be attacked by passing the file in one direction, and the where-used problem by passing it in the reverse direction.

Examples are shown in Appendix D after presentation of the linearization scheme and demonstration of an outline of the algorithm for some of the processing. The figures accompanying the examples show that the structures involved can look quite complex.(D)

A few side-issues might be noted here:

1. This discussion is directly relevant to the current Federal office-building program (where can we build what kind of office buildings?)
2. It is curious to note that one representation method for the needed information is identical to one for threshold functions.(11)

Other Applications

It should be noted that though many natural phenomena are described recursively that every realization of them ultimately turns out to be finite. Thus, so far as we are concerned, it may be possible to bring many apparently untractable problem formulations under the yoke of linearization, and hence into the domain of efficacy of the ILLIAC IV. No further detailing of such problems will be attempted here. It is sufficient at this time to know that the set of commercial applications of the forced staticization type is not empty.

Intra-Record Structure

The preceding portions of this treatise have been concerned with inter-record structure. This is because the peculiar structure of the ILLIAC IV data transfer hardware causes some unusual results and has some unpleasant consequences upon file structure. We have seen that this is due to the fact that, even though 16 PE's can decide which data blocks they each want, the reading of any data block will affect all 16 PE's, and hence they cannot really operate in parallel (i.e. independently).

Inside the record, once the data is in core, the existence of RGX, since it allows each PE to access different relative locations in its own memory, allows processing to proceed in much the manner we might expect, viz, all 64 processors can function independently. Thus, so long as pointers are confined to point within the record in which they occur, any structure which can be represented by pointers can be handled by the ILLIAC IV. Specifically, if a set of records with a full-blown network intra-record structure is subjected to a complicated query relevant to the entire file (implying that a serial pass is efficient), then the ILLIAC IV can achieve its theoretical maximum gain over other machines, namely two orders of magnitude. This covers virtually all current commercial data processing applications (payroll, billing, personnel search, etc.).

Conclusions

The ILLIAC IV machine and its associated disc are extremely fast devices. On a serial file pass they can be expected to show a gain of a factor of 200 over ordinary machines. This implies a pass of a tape-sized file in approximately 2 seconds. This has commercial value.

The simplest types of traditional improvement over serial access, namely indexed-sequential and random (hashed), were found to be operable at approximately one order of magnitude faster than predecessor machines. With more sophisticated structures the advantage seems to be lost altogether. Previous investigations by D. H. Lawrie agree with this pessimistic valuation.[3] With a gain of two orders of magnitude as a potential

target, and considering the cost of operating the ILLIAC IV, it seems that single-order-of-magnitude gains would be appropriately relegated to a secondary role.

Due to the incredibly fast serial access speed of the ILLIAC IV, ways were sought to linearize data structures so that they might then be serially accessed. Certain structures were found to be amenable to this procedure, both the structures and procedures being detailed herein.

Lastly, we have shown that both of these solutions, serial access and structural linearization, have value in that they do have actual application to real-world commercial problems.

Recommendations

This investigation has necessarily dealt with broad considerations and with qualitative results more so than quantitative ones. This is because this approach offered the greatest return for time and money invested, and was in addition an unquestionably necessary first step in the procedure of resolving the information retrieval question as it pertains to the ILLIAC IV. Hence the recommendations herein are necessarily general.

1. Attack problems where serial access is efficient. This means large scale problems of data reduction such as whole-file summaries or cross-correlations. A very large set of problems falls into this class including almost all current data processing. Also, data compaction problems are of this nature since by definition they demand that every record be examined.
2. Attack problems with linearizable structures. The class of problems encompassed by consideration of these structures is unknown but it is demonstrably not empty.
3. Do not side with the previous pessimistic approach that information retrieval *in toto* is not a suitable pasttime for the ILLIAC IV. Already enough has been seen to demonstrate that most of the current problems in the commercial area are ready grist for the ILLIAC IV.

4. CHANGE THE INFERNAL IO STRUCTURE!!!

The peculiar IO structure of the ILLIAC IV is the single and sole source of loss of efficacy with regard to any type of information retrieval problem. If the IO structure were changed to reflect the parallellism of the machine itself, great gains could be realized. These are difficult to catalogue since they are so far-reaching, but the following can be seen immediately:

A. The value of traditional methods and knowledge will be revivified. This has a host of lovable ramifications. Once traditional knowledge is applicable no wheels will have to be reinvented, a great body of literature and knowledge will become relevant, all problems and structures currently known to be solvable and their methods of solution will be immediately transferable (possibly including some that were never practical before), a speed-up of two orders of magnitude will be available on all known data structures, and finally, any advances and discoveries made by people working with the ILLIAC IV will be composed of theoretics which are directly transferable to the rest of the industry. Thus, the ILLIAC IV will be in line to reap and also to distribute benefits of knowledge and technique, instead of suffering from the need to take great pains in recreating, in its own alien way, and in an inconvenient form, all prior knowledge, let alone new concepts.

B. All the above applies equally to multi-programming and time-sharing.

C. Loss due to extended programming time, including direct loss due to greater implementation time plus indirect cost due to delay of getting a new system "on the air" will be reduced when the programmer can concentrate on the exploitation of parallellism instead of worrying about how to organize and get at the data.

D. It would pave the way for the high-powered ILLIAC IV and its high-powered associated staff to attack the single largest problem of both commerce and research, in fact the problem, data management and information systems.

E. With a suitable change in IO structure, costs due to consulting would go down, and more consultants with useful knowledge would exist. More literature and less abstruse problems would allow for a greater choice of consultants and less frequent need to utilize them. Also, since the problems would not be unique, other institutions would be working on them as well, and knowledge rather than genius would be useful in attacking succeeding problem areas, thus, evading the need for ever more ingenious and costly consulting help. As an immediate example of saved consulting costs this report can be cited; without the strange IO structure of the ILLIAC IV it would have been totally unnecessary.

APPENDIX A

Derivation of Binary Search Time Formula

If a value is to be checked against a list of p values (a list of length p) by a binary search, the maximum number of comparisons necessary to determine whether or not the value is on the list or not is:

$$[\log_2 p] + 1.$$

Hence the maximum time to make the determination is:

$$([\log_2 p]+1)*t.$$

The maximum time will be needed whenever the value is not on the list, since all the comparisons will have to be attempted; however, when the value is on the list, assume on the average only half the comparisons are necessary (specious, but it will do). Then the probable time, T, to find the value or learn that it is not present is:

T = Prob(value not on list)* maximum-time
+ Prob(value on list) * 1/2 * maximum-time.

If there are n possible values from which the p values on the list were selected, then:

$$T = ((1-p)*([\log_2 p]+1)*t) + (\frac{1}{2}*\frac{p}{n}*([\log_2 p]+1)*t)$$

or $$T = (1-\frac{p}{2n}) * ([\log_2 p]+1) * t.$$

If a buffer is to have all the records it holds checked against such a list and the buffer is contained in a single PEM, then the total time to check that buffer depends on the number of records in that buffer, [b/d]:

$$T = [b/d]*(1-\frac{p}{2n})*([\log_2 p]+1)*t.$$

T is the time it takes each PE to process the records in its PEM against the list of requested records.

APPENDIX B

Index-Sequential Access Structure

Index-sequential is a file organization in which logical data records are ordered ascendingly (the sequential aspect) with respect to the value of some field (the key) in each record. Logical records are packed into physical blocks. An ordered list is kept of the highest value in each block (the index aspect); associated with each such value is a pointer to the appropriate block. Each such pair of values (key, pointer) is considered to be a logical index record. These in turn are packed into physical blocks. If more than one such block exists, the index blocks are themselves indexed, that is, an ordered list of the highest key value in each block (and the associated pointer) is kept. Clearly if r index records can fit in a physical block, this second index list will be, at most, $\frac{1}{r}$th the size of the previous ones. This list is in turn blocked and indexed, and the process continues until the list can be contained in a single block. This must happen, since $(1/r)^n$ approaches 0 as n gets large.

The index-sequential method, then, clearly depends on a set of primary (i.e. determining physical order) keys and associated pointers. We already know that on the ILLIAC IV all pointer methods are doomed, and consequently that the ILLIAC IV will not be adequate to the index-sequential task, or at least, not nearly so much so as we would like.

APPENDIX C

Calculation of Expected Number of Utilized Index Blocks

The classical probability theory analogue to determining how many of the n_1 (64) index blocks are needed to access the p (100) records is the "occupancy problem" which deals with distributing p balls into n urns. Under the usual assumptions of lack of bias (each urn is an equally likely receptacle for each ball), the expected number of occupied urns is: (10)

$$E = n * (1-(1-(1/n))^p)$$

Consequently, the expected number of index blocks needed by the 100 batched requests is:

$E = 2^6*(1-(1-(1/2^6))^p) =$ 51 index blocks (out of 64)

$E = 2^4*(1-(1-(1/2^4))^{100} =$ 16 index blocks (out of 16) $\quad (\frac{15}{16})^{100} \simeq \emptyset$

$E = 10^6*(1-(1-(1/10^6))^{100}) =$ 100 data blocks.

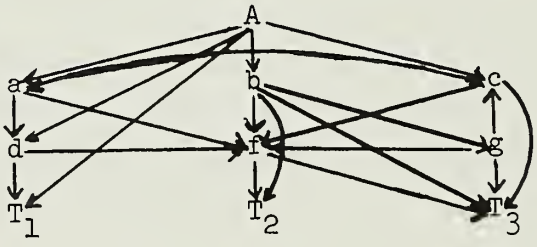
APPENDIX D

The Linearization Scheme

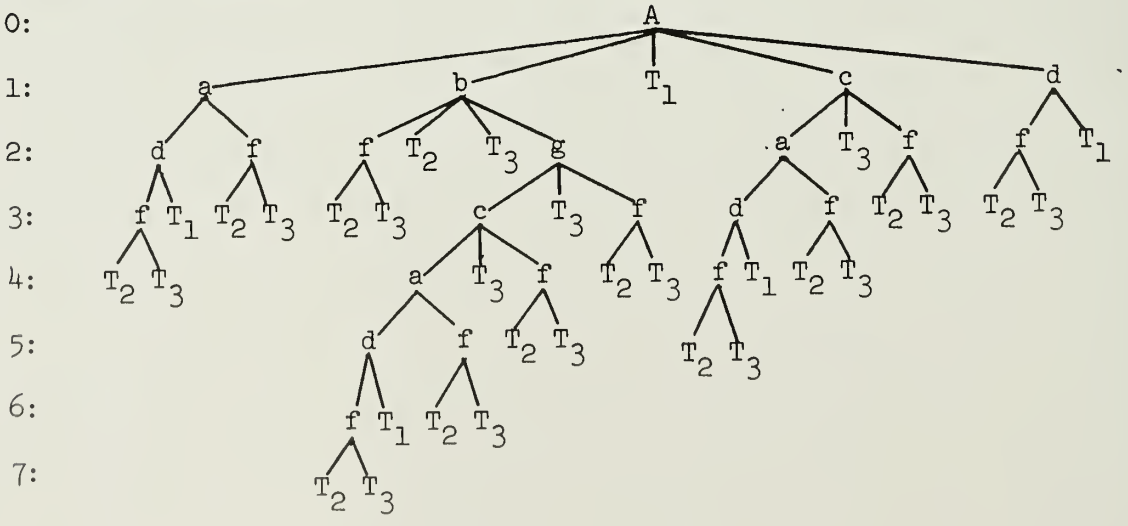
The following is an outline of how to go about linearizing a graph in the suggested manner.

1. Each node with no fan-in is expanded into a tree.
2. The levels are numbered and each participating element is associated with its maximum level number. Nodes with no fan-out may be demoted to maximum level. (Also, if necessary, nodes with fan-out only to maximum level nodes may be demoted to maximum-1 level, etc.)
3. The records representing each node are then ordered ascendingly on the assigned numbers. If two elements have the same number then their ordering with respect to each other is irrelevant (denoted by enclosure within parentheses - excluding this pair.)

Example 1: The graph of a single assembly.



Expansion to a tree:



The ordering of the records is now seen to be

A b g c a d (f T_1) (T_2 T_3)

or, demoting T_1:

A b g c a d f (T_1 T_2 T_3).

The connecting links between graph nodes can represent more than simply connections. Numbers can be placed on them to represent distance, costs, or other relational phenomena. In fact these numbers often give rise to matrices such as linear combinations in Operations Research. This topic deserves some expansion of its own, but not here.

For the purpose of our examples, the numbers carried by the arrows (though not actually shown there) will represent the number of subordinate components of each type needed to construct the superior node, i.e., one instance of the assembly represented by the superior node. These numbers will be illustrated in the records representing the file which demonstrates the linearization of our example graphs. The records will be shown in the form:

IDENTIFIER : <number identifier> ...

with all descriptive data pertaining to the particular component omitted from our sample records.

The file organization of Example 1 is then:

A: 2a 3b 4c 2d $5T_1$
b: 3f 2g $4T_2$ $2T_3$
g: 2c 3f $4T_3$
c: 1a 2f $3T_3$
a: 1d 2f
d: 2f $5T_1$
f: $3T_2$ $2T_3$
T_1:
T_2:
T_3:
EOF (End of File).

For example, then, a is seen to be composed of one d and 2 f's.

We now consider the problem of discovering just how much of everything we need to make one subassembly b.

The totaling algorithm:

1. Establish an accumulator for each relevant component.
2. When a component which has an accumulator is read, multiply its component definition numbers by the value of its accumulator and accumulate those numbers to the proper accumulators (if a component is irrelevant it may be thought to have a value of 0).

The process is started by establishing an accumulator. It is demonstrated in detail below:

Step	Record	Accumulators
Init	-	b=1 (find components needed to make 1 b)
1	A	b=1 (A is not relevant, no accumulator, ignore it)
2	b	b=1: f=1*3 g=1*2 T_2=1*4 T_3=1*2 (mult by b=1)
3	g	b=1 g=2 (mult by g=2)
		(b=1) g=2: c=0+2*2 f=3+2*3 T_2=4 T_3=2+2*4 or
		(b=1) g=2: c=4 f=9 T_2=4 T_3=10
4	c	(b=1 g=2) c=4: a=0+4*1 f=9+4*2 T_2=4 T_3=10+4*3 or
		(b=1 g=2) c=4: a=4 f=17 T_2=4 T_3=22
5	a	(b=1 g=2 c=4) a=4: d=0+4*1 f=17+4*2=25 T_2=4 T_3=22
6	d	(b=1 g=2 c=4 a=4) d=4: f=25+4*2=33 T_1=0+4*5=20 T_2=4 T_3=22
7	f	(b=1 g=2 c=4 a=4 d=4)f=33: T_1=20 T_2=4+33*3=103 T_3=22+33*3=88
8-10	T_1-T_3	b=1 g=2 c=4 a=4 d=4 f=33 T_1=20 T_2=103 T_3=88

Therefore, an extended record of b describing its total composition, would be:

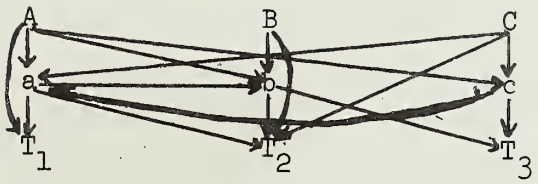
b: 2g 4a 4c 4d 33f 20T_1 103T_2 88T_3 (excepting arithmetic errors, that is.)

To compute the proper number of all types of components needed to construct five A's, one would begin by initializing an accumulator for A to 5 (A=5).

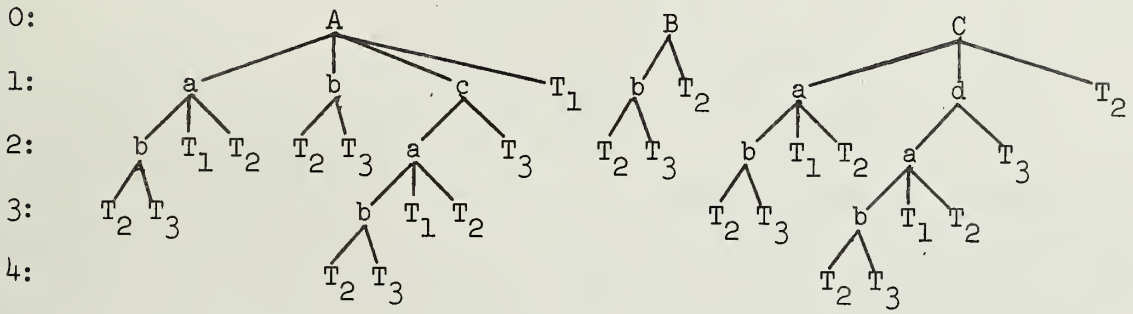
In the next example we consider a few trees woven together, though each is less complex than in Example 1 to avoid obscuring the content. Both the totaling process and the where-used accumulation process will be shown.

Example 2

Graph:



Expansion to trees:



The orderings are then seen to be:

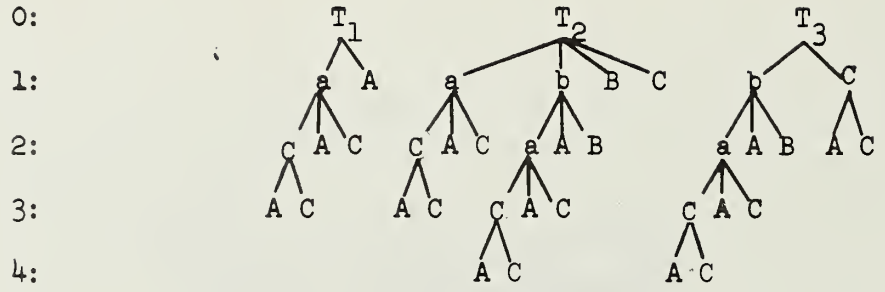
A: c a (b T_1) (T_2 T_3)

B: b (T_2 T_3)

C: c a (b T_1) (T_2 T_3)

The combined ordering, taking all three trees as a single graph, or using the common ordering defined by the above (and the demotion rule for terminals) gives: (A B C) c a b (T_1 T_2 T_3)

To demonstrate the where-used chain, we recreate the trees from the ground up, assuming that all the arrows in the graph point in the reverse direction from that in which they are actually depicted.



Thus the orderings are seen to be:

T_1: a c (A C)

T_2: b (a B) c (A C) equivalently: b a c (A B C)

T_3: b (a B) c (A C) equivalent as above (demotion of terminals rule).

The combined ordering is then: (T_1 T_2 T_3) b a c (A B C).

This ordering is exactly the reverse of the explosion ordering, as expected. Consequently, the explosion question can be answered by a forward file pass and the where-used question by a backwards file pass. We demonstrate the file, again including only the composition information but none of the other (descriptive) data which may occur in each record, such as vendor, characteristics, price, etc. The format is as before with the addition of "//" to separate the explosion chain from the where-used chain and "*" to terminate a record.

File:

A: 2a 3b 4c 5T_1 //*

B: 2b 3T_2 //*

C: 2a 3c 2T_2 //*

c: 1a 3T_3 // A C *

a: 2b 4T_1 3T_2 // c A C *

b: 2T_2 3T_3 // a A B *

T_1: // a A *

T_2: // a b B C *

T_3: // b c *

EOF (End of File)

To find the total for assembly C: (say 3 assembly C's)

Step	Record	Accumulators
Init	-	C=3
1-2	A,B	C=3
3	C	C=3: a=0+3*2=6 c=0+3*3=9 T_2=0+3*2=6
4	c	(C=3) c=9: a=6+9*1=15 T_2=6 T_3=0+9*3=27
5	a	(C=3 c=9) a=15: b=0+15*2=30 T_1=0+15*4=60 T_2=6+15*3=51 T_3=27
6	b	(C=3 c=9 a=15) b=30: T_1=60 T_2=51+30*2=111 T_3=27+30*3=117
7-9	T_1-T_3	C=3 c=9 a=15 b=30 T_1=60 T_2=111 T_3=117

Thus a super-record for C is C: 3c 5a 10b 20T_1 37T_2 39T_3

Now we consider an algorithm for a where-used expansion:

1. For every relevant record found, annex its where-used list to an overall where-used list.
2. For every name on the overall where-used list find its record and take a linear combination of accumulators with its composition.

The process is initialized by establishing a counter for the desired component with a value of 1.

Where-used expansion for T_2 (recall the file is read backwards) is:

Step	Record	Accumulators and list
Init	-	T_2=1
1	T_3	T_2=1 (T_3 irrelevant, has no counter)
2	T_2	T_2=1 // a b B C (Note T_2 was considered on the list)
3	T_1	T_2=1 // a b B C (T_1 has no counter, irrelevant)
4	b	T_2=1 b=1*2=2 // a B C (b=2*T_2+3*T_3; T_2=1 T_3=0)
5	a	T_2=1 b=2 a=2*2+3*1=7 // c A B C
6	c	T_2=1 b=2 a=7 c=1*7=7 // A B C
7	C	T_2=1 b=2 a=7 c=7 C=2*7+3*7+2*1=37 // A B
8	B	T_2=1 b=2 a=7 c=7 C=37 B=2*2=4 // A
9	A	T_2=1 b=2 a=7 c=7 C=37 B=4 A=2*7+3*2+4*7=53

Thus a super where-used record, describing how much of component T_2 is required to construct various assemblies might be:

T_2: 7a 2b 7c 53A 4B 37C.

Note the figure for C: 37. One C can be made using 37 T_2's. This agrees with the expansion of C given in this example: one of the components of C is T_2 and 37 of them are needed.

REFERENCES

1. Lawrie, Duncan H. "A Preliminary Study of Information Retrieval on ILLIAC IV," ILLIAC IV Document No. 94 and "A Further Note on Information Retrieval," ILLIAC IV Document No. 96 (formerly numbered ILLIAC IV #148 and ILLIAC IV #150, respectively). ILLIAC IV Project, University of Illinois at Urbana-Champaign.

2. Lawrie, Duncan H. "A Preliminary Study of Information Retrieval on ILLIAC IV," ILLIAC IV Document No. 94. ILLIAC IV Project, University of Illinois at Urbana-Champaign. Page 1.

3. Lawrie, Duncan H. "A Further Note on Information Retrieval," ILLIAC IV Document No. 96. ILLIAC IV Project, University of Illinois at Urbana-Champaign. Page 6. (Actually, this is the last sentence of ILLIAC IV Document No. 96, and therefore, quite likely Lawrie's final word - literally - and opinion on the subject.)

4. Schuster, Stewart A. "An Information Management and Analysis System for ILLIAC IV," CAC Document No. 1. Center for Advanced Computation, University of Illinois at Urbana-Champaign. Pages 8 and 9. (It should be noted here that Mr. Schuster's time estimates which appear on page 8 of CAC #1 disagree with the author of this document's estimates solely because Mr. Schuster's foundation has been outdated. His diagram on page 7 of CAC #1 is quite useful and has been reproduced herein as Figure 1.)

5. McIntyre, David E. Telephone conversation, 24 September 1971. ILLIAC IV Project, University of Illinois at Urbana-Champaign.

6. Schuster, Stewart A. "An Information Management and Analysis System for ILLIAC IV," CAC Document No. 1. Page 7.

7. Graham, Marvin L. Consultation, 16 September 1971. ILLIAC IV Project, University of Illinois at Urbana-Champaign.

8. Schuster, Stewart A. "The Tuning of Buffer Parameters for the ILLIAC IV Data Management System," CAC Document No. 3. Page 9.

9. Alsberg, Peter A. Consultation, 23 September 1971. Center for Advanced Computation, University of Illinois at Urbana-Champaign.

10. Parzen, Emanuel. Modern Probability and Its Applications, LC-#60-6456. John Wiley and Sons, New York. 1960. Page 368.

11. Wittman, Barry and Peter Z. Ingerman. "A Threshold Selection Language," Proceedings of the 20th National ACM Conference. 1967.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

AN INVESTIGATION INTO INFORMATION RETRIEVAL UTILIZING ILLIAC IV

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

5. AUTHOR(S) (First name, middle initial, last name)

B. Wittman

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
November 30, 1971	33	11

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAHC04 72-C-0001 b. PROJECT NO. ARPA Order 1899 c. d.	CAC Document No. 22 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Copies may be obtained from the address given in (1) above.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	U. S. Army Research Office-Durham Duke Station Durham, North Carolina

13. ABSTRACT

This document presents an investigation into the effective use of the ILLIAC IV for information retrieval. It notes the lack of parallelism in the Io structure and finds two orders of magnitude gain over serial machines when using serial processing. It recommends serial access methods and linearized structures where such structures are possible. It also recommends changing the ILLIAC IV IO structure. Further, the document demonstrates a linearization technique for graphs.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Information Retrieval (General)						
Evaluation of Systems						
Searching						
Data Structures						